Vili Raassina

Testaustulosten tallentaminen tietokantaan

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Automaatiotekniikka

Insinöörityö

21.2.2014

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Vili Raassina Testaustulosten tallentaminen tietokantaan 35 sivua + 1 liitettä 21.2.2014
Tutkinto	insinööri (AMK)
Koulutusohjelma	Automaatiotekniikka
Suuntautumisvaihtoehto	
Ohjaaja(t)	Jani Angervuo, Chief Design Engineer, Production Testing Antti Liljaniemi, Lehtori

Tämän opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa testaustulosten tallentaminen tietokantaan tiedostojen paikallisen tallentamisen päivittämiseksi. Työssä suunniteltiin ohjelma, jonka tarkoituksena oli tehdä tiedostoille parsinta, tallentaa määritellyt tulokset tietokantaan ja suorittaa käsitellyille tiedostoille tarvittavat jatkotoimenpiteet. Tavoitteena oli toteuttaa täysin automaattinen kokonaisratkaisu, joka voidaan asentaa sijoittamalla tiedosto testerikoneelle, määriteltyyn polkuun.

Työhön liittyvää teoriaosuutta selvitettiin kirjallisuuden ja verkkodokumenttien avulla, joista saatiin tarvittava tieto työn suunnitteluun. Tietokanta- ja ohjelmistosuunnittelun pohjalta rakennettiin ohjelma, joka suorittaa suunnitellut toiminnot.

Työssä käytettiin avuksi CMMI-ohjelmistotuotannon laadunhallintamenetelmää, joka kattaa ohjelmistoprojektille ominaisen elinkaarimallin.

Työtä tehdessä havaittiin testaustiedostojen formaattiin liittyviä ongelmakohtia, mikä on tärkeä osa työtä. Näistä havainnoista laadittiin kokoon tarvittavat tiedot korjaustoimenpiteitä varten.

Työssä havaittiin ja opittiin ohjelmistoprojektille ominaisia piirteitä, joita ovat esimerkiksi eri testauksien liittyminen oleellisena osana toteutukseen.

Lopputuloksena saatiin vaatimusmääritysten mukainen ohjelma ja testaustiedostojen formaattiin liittyvä dokumentti, joka auttaa ratkaisemaan testereihin liittyviä ongelmakohtia.

Työ tehtiin hissi- ja liukuporrasyhtiö KONEen R&D-osastolle Hyvinkäällä.

Avainsanat	Ohjelmistoprojekti, tietokanta, XML

Author(s) Title Number of Pages Date	Vili Raassina Storing Test Results to a Database 35 pages + 1 appendices 21 February 2014
Degree	Bachelor of Engineering
Degree Programme	Automation engineering
Specialisation option	
Instructor(s)	Jani Angervuo, Chief Design Engineer, Production Testing Antti Liljaniemi, Principal Lecturer

The purpose of this thesis was to design and implement a system that stores test results to a database. A software was planned to parse files, store results to database and perform needed actions of existing files. Target was to implement a fully automated system solution which can be installed to tester machine in defined path.

Theory of project was investigated with literature and web documents where the needed information was gotten for the design process. Based on database and software design a defined program was created.

In this project CMMI software engineering quality management method that cover life-cycle model of software project was used.

Important part of project was detecting XML files format challenges. Based on these detections an information document for needed corrective actions was created.

In project features of software project was noticed and learned.

The end result was requirement specification software and document of testing files format, which help to solve crucial challenge points.

Project was done for KONE Corporation Research and Development Department in Hyvinkää.

Keywords	Software project, database, XML

Sisällys

Lyhenteet

TKHJ	Tietokannan hallintajärjestelmä. Ohjelmisto, jonka avulla hallinnoidaan tietokantoja.
XML	Extensible Markup Language. Tiedon merkintä standardi.
W3C	World Wide Web Consortium. Ylläpitää ja kehittää www:n standardeja.
SOAP	Simple Object Access Protocol. Tietoliikenne protokolla.
SQL	Structured Query Language. Standardoitu kyselykieli.
PHP	Hypertext Preprocessor. Ohjelmointikieli.
UTF-8	Unicoden merkistöstandardi.
IT	Informaatioteknologia
MS	Microsoft. Ohjelmistoyritys.
OSI	Tiedonsiirron standardi
WebService	Tiedonvälitysprotokolla internetin yli.
IP	Internet Protocol. Internet-kerroksen protokolla.
PDCA	Plan-Do-Check-Act. Laadunohjausprosessi.
CMMI	Capability Maturity Model Integrated. Laadunhallintaprosessi.
XSTL	Extensible Stylesheet Language Transformations. Merkintäkieli.
XPath	XML-kieli
Triggeri	Liipaisin, tehtävien aloitustekijä.
SNMP	Simple Network Management Protocol. Tietoliikenneprotokolla.

HTML	Hypertext Markup Language. Kuvauskieli.

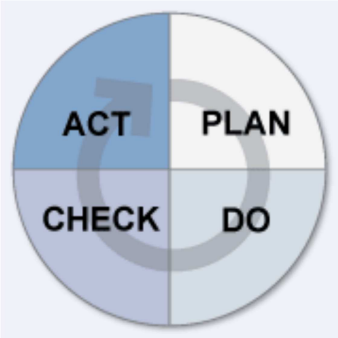
1 Johdanto

1.1 Opinnäytetyö

Opinnäytetyön aiheena on elektroniikan tuotannossa olevien tuotantotestereiden testaustulosten tallentaminen tietokantaan. Tavoitteena on antaa lukijalle tarvittava tieto työhön liittyvästä teoriasta sekä toteutuksesta, jonka avulla voidaan lähteä joko haastamaan tai kehittämään samankaltaista toteutusta.

Työssä käytetään laadunvarmistamiseksi CMMI (Capability Maturity Model Integrated) -ohjelmistotuotannon laadunhallintamenetelmää, joka käsittää kuvan 1 mukaisen ohjelmistotuotannon elinkaarimallin ja prosessiin sisältyy kuvan 2 mukainen PDCA-malli (Plan-Do-Check-Act), jota voidaan soveltaa lähtökohtaisesti kaikissa prosesseissa. Ohjelmistotuotannon työmäärä jakautuu tyypillisesti 10 % vaatimuksiin, 10 % suunnitteluun, 10 % koodaukseen, 20 % testaukseen ja 40 - 80 % ylläpitoon.

Kuva 1. Ohjelmistotuotannon laadunhallinnan elinkaarimalli. (1, s. 26.)



Kuva 2. PDCA-malli. (2, s. 45.)

Työtä lähdetään suorittamaan tutkimalla työhön liittyvien osien teoriapuolta, joka tukee toteutuksen onnistumista ja ymmärtämistä. Teoriaosuuden jälkeen määritellään vaatimukset, joiden pohjalta suunnitellaan toteutuksen vaiheet. Näiden alkutekijöiden tuloksena saadaan tarvittava tieto työn suorittamisesta. Lopuksi arvioidaan työn onnistuminen ja pohditaan työssä esille tulleita asioita.

Testiympäristö rakennetaan paikallisesti testaustietokoneelle ja projektin onnistuminen voidaan määrittää tulevasta testausympäristön toiminnasta sekä hyödynnettävyydestä tuotantotesterille. Toteutus sisältää tietokannan suunnittelun sekä sinne kirjoittavan ohjelmiston suunnittelun. Ohjelman tulee toimia täysin automaattisesti vaatimusmäärittelyn mukaisesti.

Työ suoritetaan hissi- ja liukuporrasyhtiö KONE Oyj:n tuotekehitysyksikölle tarkoituksena tuotantotestereiden päivittämisen tukeminen. Työ antaa yritykselle tietoa käytetyistä ohjelmistoista, sekä niiden hyödynnettävyydestä tulevissa tuotekehitysprojekteissa ja mahdollisuuden käyttää hyödyksi toteutettua ohjelmaa tuotannon testereille.

1.2 KONE Oyj

KONE Oyj tarjoaa asiakkailleen People Flow ratkaisuja ja on keskittynyt tehokkaaseen sekä joustavaan henkilöiden liikkuvuuteen erilaisissa kaupungistumisen ympäristösovelluksissa. Kehittyneitä ja kilpailukykyisiä tuotteita ovat hissit, liukuportaat, liukukäytävät sekä automaattiovet, joilla henkilövirtaa voidaan tehokkaasti ohjata rakennusten sisätiloissa. (3, KONE lyhyesti.)

KONE vaikuttaa noin 50 maassa eri puolilla maailmaa ja sillä on seitsemän eri tuotantoaluetta päämarkkinoilla sekä kahdeksan tutkimus- ja tuotekehityskeskusta. Tutkimus- ja tuotekehityskeskukset sijaitsevat Hyvinkäällä, Kunshanissa, Chennaissa, Essenissä, Cadrezzatessa, Torreonissa, Allenissa ja Coal Valleyssä. Hyvinkäällä sijaitsee KONEen tuotantotehdas, jossa valmistetaan erikoishissejä asiakkaan tarpeiden mukaan ja tuotekehitysyksikkö. Tuotekehitysyksikköön kuuluu luotettavuuslaboratorio, jossa kehitetään, testataan sekä tarkistetaan yhtiön tarjoamia tuotteita. Yrityksen pääkonttori sijaitsee Espoon Keilaniemessä. (3, KONE lyhyesti.)

KONE tarjoaa kokonaisvaltaisia hissi-, liukuporras-, automaattiovi- ja liukukäytäväratkaisuja, sekä modernisointi ja kunnossapitopalveluita. (3, KONE lyhyesti.)

Testaus on edellytyksenä laadukkaille ja luotettaville tuotteille, sekä tuotekehitykselle kannattavana tietolähteenä. Tuotekehityksen kannalta hyödylliset testitiedot antavat hyvät mahdollisuudet korjaaviin optimointitoimenpiteisiin, jotka edesauttavat tuotteiden laatua ja luotettavuutta. Tietokantaan tallennetut testitiedot mahdollistavat suurien tietomäärien analysoinnin sekä testaustulosten tutkimisen yksittäiselle moduulille.

2 Tietojenkäsittelyyn vaikuttavia tekijöitä

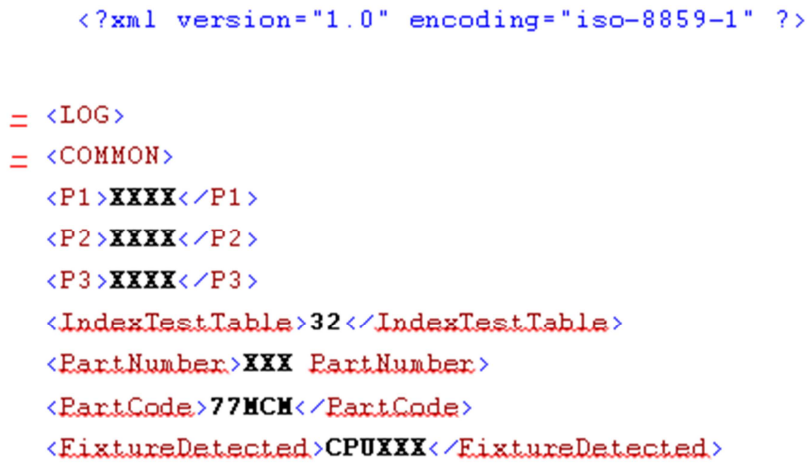
2.1 Tiedon tallentaminen

Tiedot tallennetaan kovalevylle, tietue-, binääri- tai tekstiformaatissa sovellusten tuottaessa laskentataulukoita, osoitekirjoja tai rakenteellista dataa. Tekstiformaatin etu on sen luettavuudessa, mikä mahdollistaa tekstin lukemisen ilman kyseisen tekstin tuottaneen ohjelman käyttämistä. XML-tiedostot (Extensible Markup Language) ovat aina suurempia kuin vastaavat binääritiedostot, joten tekstiformaatin valinta on yleensä tietoisesti harkittu päätös, koska edut ovat paremmat. Lisäksi pakkausohjelmilla saa tiedostoja pakattua tiiviimmäksi ja eri protokollat voivat pakata dataa käyttäen tiedonsiirtoa yhtä tehokkaasti kuin binääriformaatitkin. XML ja sen käyttötavat ovat lähes aina erilaisia, joten saavutettavat hyödyt riippuvat suunnittelu- ja toteutustyön onnistumisesta. (4)

2.2 XML-standardi

Kun mainitaan XML-termi, se ei viittaa suoraan tiettyyn kieleen tai muotoon, vaan se koostuu eri teknologioista, jotka ovat W3C:n kehittämiä. Päätavoitteena on välittää viestejä järjestelmien välillä. Tämänlaisia XML-standardia hyödyntäviä järjestelmiä ovat esimerkiksi WebServices ja SOAP (Simple Object Access Protocol). (5, XML Basic; 6, XML)

XML-kielen päätavoitteet ovat yksinkertaisuus, yleistyneisyys sekä käytettävyys tietoverkoissa. Tarkoituksena kielelle on laitteisto- ja ohjelmistoriippumattomuus. Dokumentti koostuu elementeistä ja alkaa prologilla, johon on sisällytetty versio, dokumentin koodaustietoja ja muita tietoja. Kuvauskieli perustuu tagien ’<’ ja ’>’ ja attripuuttien nimi=’arvo’ käyttöön ja se käyttää tageja vain tekstidatan rajaamiseen ja jättää tulkinnan tietoa käsittelevälle sovellukselle. Esimerkkitiedoston kuvan 3 mukaan prologin jälkeen tulee seuraavaksi tiedoston aloituselementti ”<LOG>” ja tiedoston lopetuselementti ”</LOG>”, tiedoston lopussa. Seuraavaksi tulee ”<COMMON>” elementti, jonka alla kerrotaan testin yleistietoja. (5, XML Basic; 6, XML)



Kuva 3. Ote XML-tiedostosta.

Jokaisessa tiedostossa tulee olla yksi juurielementti, jolla on aloitus- ja lopetustagit, joka kuvassa 3 esitetyssä tiedostossa on elementti nimeltä LOG. Elementin sisällä aloitettuja elementtejä kutsutaan lapsielementeiksi, eli kuvassa 3 COMMON elementti on LOG elementin lapsielementti ja P1 on COMMON elementin lapsielementti. Elementit nimetään isäntä-, lapsi-, sisaruselementeiksi riippuen siitä mistä näkökulmasta asiaa havainnoidaan. Kaikki elementit suljetaan </elementin nimi> -merkeillä. (5, XML Basic; 6, XML)

Elementtien nimi- ja arvokentillä on rajoituksia, jotka tulee ottaa huomioon. Tämänkaltaisia asioita ovat elementin nimen alkaminen ainoastaan kirjaimella, alleviivalla tai kaksoispisteellä. Nimi- ja arvokentät saavat sisältää kaikkia muita merkkejä paitsi vertailumerkkejä suurempi kuin ja pienempi kuin -merkkejä, sekä *&*-, heitto- tai lainausmerkkejä. Kyseisten merkkien tilalle on määritelty samaa tarkoittavat merkit, joita ovat *&gt* suurempi kuin, *&lt* pienempi kuin, *&amp* &, *&apos* heitto ja *&quot* lainaus. Tiedostoon voidaan sisällyttää myös kommentteja, jotka sijoitetaan erilliseen elementtiin <!-- ”kommentti” -->. Elementteihin voidaan määritellä muuttujia, joiden arvot tulee olla lainausmerkeissä. Samannimisiä muuttujia ei saa olla toiseen kertaan yhdessä elementissä ja muuttujien nimeämisessä huomioidaan samat merkkisäännöt kuin elementti- ja arvokenttien nimeämisessä. (5, XML Basic; 6, XML)

XML:ssä voidaan käyttää ohjelmoinnin yhteydestä tuttuja nimiavaruuksia. Nimet voidaan määrittää eri nimiavaruuksiin, joten jos olisi kaksi saman nimistä silti ne voidaan yksilöidä, jos ne sijoitetaan eri nimiavaruuksiin. Nimiavaruuden määrittäminen suoritetaan kaksoispisteellä ja kaksoispisteen etupuolella oleva nimi määrittää avaruuden, esimerkiksi *ms:tietokanta*. Näin ollen tietokanta nimeä voidaan hyödyntää myös muissa avaruuksissa, kun viitataan tietokannan kehittäjän lyhenteeseen niin kuin esimerkissä Microsoftiin. (5, XML Basic; 6, XML)

XML DTD

DTD (Document Type Definition) on malli rakenteellisten dokumenttien kuvaukseen. DTD:n avulla voidaan kuvata XML-dokumentin rakennetta ja se toimii vahvistamismenetelmänä dokumenteille, joka hyväksyy tai hylkää tiedoston. DTD on kehitetty ennen XML:ää, joten ei pidä sekoittaa niitä keskenään, sitä käytetään vain hyväksi vahvistamaan tiedoston muoto. Kuvassa 4 ensimmäisellä ELEMENT rivillä esitellään note-niminen elementti, joka koostuu to-, from-, heading-, body-lapsielementeistä. Seuraavilla neljällä rivillä jokainen lapsielementti esitellään erikseen ja kerrotaan määritelty tyyppi. (5, XML Basic; 6, XML)

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

Kuva 4. DTD määrittelydokumentin rakenne. (5, XML Basic)

Muuttujan tyyppi kerrotaan sulkujen sisällä, jonka eri vaihtoehdot on listattu taulukkoon 1. Muuttujien arvojen määritykseen on olemassa omat tyyppinimet, ja tiedostossa pystytään määrittelemään myös olioita. (5, XML Basic; 6, XML)

Taulukko 1. Eri tyyppivaihtoehdot.

Tyyppi	Määrittely
CDATA	Sisältää tekstidataa
(en1|en2|..)	Arvo tulee olla jokin määritellyistä
ID	Arvo tulee olla yksilöllinen ID:llä merkityissä muuttujissa
IDREF	Arvo tulee olla joku ID:ksi määritelty muuttuja
IDREFS	Arvo on lista IDREF arvoista
NMTOKEN	Arvo on validi XML nimi
NMTOKENS	Arvo on lista valideista NMTOKEN arvoista
ENTITY	Arvo on erityinen arvo
ENTITIES	Arvo on lista ENTITY arvoista
NOTATION	Arvo on notaation nimi

XML Schema

XML Schema on XML-pohjainen laajennettu versio DTD:stä, jolla kuvataan XML-muotoisen datan rakennetta ja vahvistetaan tiedoston formaatti. Tiedostoa voidaan muokata millä tahansa XML-editorilla ja se on helposti laajennettavissa. Esimerkki XML Scheman muoto, koostuu xs-nimiavaruudesta ja eri arvotyypeistä kuvassa 5. (5, XML Basic; 6, XML)

```
<xs:element name="note">

<xs:complexType>
  <xs:sequence>
    <xs:element name="to" type="xs:string"/>
    <xs:element name="from" type="xs:string"/>
    <xs:element name="heading" type="xs:string"/>
    <xs:element name="body" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

</xs:element>
```

Kuva 5. XML Scheman formaatti. (5, XML Basic)

XPath

XPath on XML-kieli, jota käytetään elementtien etsimiseen tiedostosta ja se pohjautuu XML DOM-rakenteeseen. XML-tieto ladataan muistiin, jolloin sille voidaan suorittaa

toimintoja. Kieli sisältää suuren määrän komentoja, joiden avulla tietoja voidaan määritellä haettavaksi. Yleisimmät komennot on listattu taulukkoon 2, enemmän komentoja W3C:n verkkosivuilla. (5, XML Basic; 6, XML)

Taulukko 2. Yleisimmät XPath komennot.

Komento	Määrittely
elementin nimi	Valitsee kaikki nimeä vastaavat lapsielementit
/	Ensimmäisenä merkkinä valitsee juuren lähtöpisteeksi, muuten valitsee lapsielementtejä edellisestä elementistä
//	Valitsee tästä paikasta alaspäin
.	Valitsee nykyisen elementin
..	Valitsee isäntäelementin
@	Valitaan muuttuja
/elementti/lapsielementti[X]	Valitaan X lapsielementti, joka on elementin lapsielementti
/elementti/lapsielementti[last()]	Valitaan elementin lapsielementin viimeinen elementti
//*	Valitsee kaikki elementit

XML XSLT

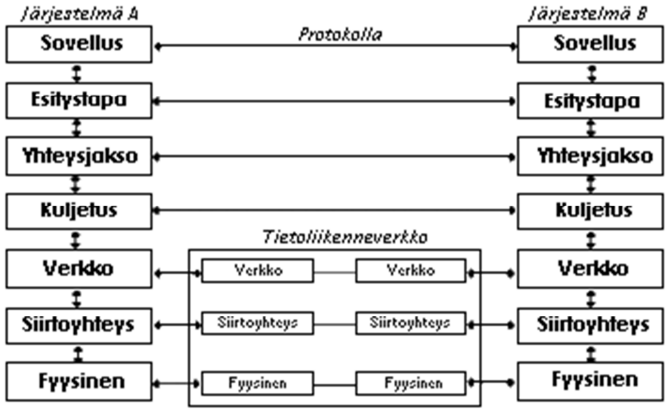
XSLT:llä (Extensible Stylesheet Language Transformations) muutetaan tiedosto johonkin muuhun muotoon. Yleensä XML-muotoinen tieto halutaan muuttaa luettavaan muotoon esimerkiksi HTML (Hypertext Markup Language) -muotoon, jota selaimet käyttävät. XHTML on HTML-kielen versio, joka on tehty puhtaaksi XML-kieleksi ja dokumentti on UTF-8-muotoista. Muutos koostuu XSL-tiedostosta ja XML-tiedostosta, joilla päästään lopputulokseen. XSL-tiedoston elementit ovat nimiavaruudessa xsl. (5, XML Basic; 6, XML)

2.3 Tiedonsiirto

Tiedonsiirto voidaan jakaa eri ryhmiin, joita ovat digitaalinen tai analoginen tiedonsiirto, yhteyspohjainen tai pakettivälitteinen tiedonsiirto. Kaikki tietokonepohjaisesti välitettävä tieto on yleensä aina digitaalista, kun taas äänet ja kuvat ovat analogista tietoa. Äänen ja kuvan saa myös pakattua digitaaliseksi. Tiedonsiirto voi tapahtua analogisesti vaikka itse tieto olisi digitaalista. Tällöin käytetään D/A- ja A/D-muuntimia, joilla tieto voidaan muuttaa digitaalisesta analogiseksi ja toisin päin. Pakettivälitteisessä tiedonsiirrossa

lähettäjä tekee paketin vastaanottajan osoitteella varustettuna ja jättää sen verkolle lähetettäväksi. (7, Kappale 19)

Kun halutaan käyttää samanlaista ideologiaa kaikkialla tukeudutaan standardeihin, joten tiedonsiirrostakin on standardi, OSI-malli (Open Systems Interconnection). OSI-malli koostuu seitsemästä kerroksesta kuvassa 6, jotka ovat alhaalta ylöspäin, *fyysinen, siirtoyhteys, verkko, kuljetus, yhteysjakso, esitystapa ja sovellus*. Malli toimii ylhäältä alaspäin eli ylempi kerros käyttää hyväkseen alempia kerroksia. (7, Kappale 19)



Kuva 6. OSI-malli. (7, Kappale 19)

Fyysinen kerros muodostuu kaikista sähköisistä, mekaanisista ja loogisista osista. Tällä kerroksella tietoa voidaan siirtää sarjamuotoisesti tai rinnakkaismuotoisesti. Sarjamuotoisen viestin etuna voidaan pitää johtimien määrää, koska niitä ei välttämättä tarvita kuin kaksi tai molemmin suuntaista tietoa varten kolme. Sarjamuotoisessa tiedonsiirrossa bitit lähetetään peräkkäin, joten tavulle määritetään aloitus ja lopetus bitit. Rinnakkaismuotoisessa tiedonsiirrossa merkin kaikki bitit lähetetään samaan aikaan, kukin omaa johdinta pitkin. Tässä tapauksessa tiedonsiirto on nopeampaa ja merkkien erottamiseen käytetään usein liipaisujohdinta. (7, Kappale 19)

Rinnakkaista tiedonsiirtoa käytetään pääasiassa tietokoneen sisällä ja tietokoneen lähellä olevien oheislaitteiden välillä, koska useiden johtimien vuoksi se ei ole kannattavaa pitkillä matkoilla. Siirtoyhteyskerros hoitaa yhteyden luomisen ja purkamisen sekä virheiden korjaamisen. Se pitää myös huolen siitä, ettei tietoa lähetetä nopeammin, kuin vastaanottaja pystyy käsittelemään. Virheettömyyttä

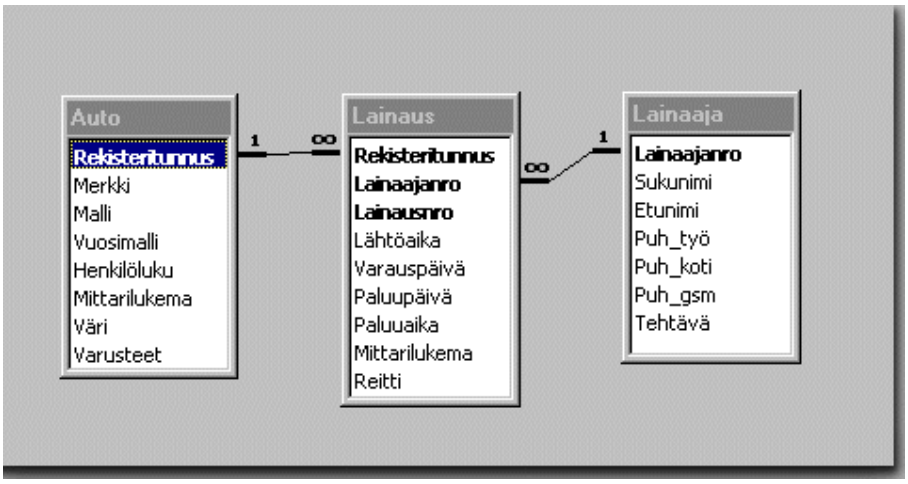
tarkastetaan käyttämällä niitä havaitsevia ohjelmia ja lähettämällä virheellinen data tieto uudelleen. Verkkokerroksen tarkoituksena on antaa verkon rakenteesta riippumaton tiedonsiirto. Ajatuksena tälle kerrokselle on, että samanlaisen verkkokerroksen omaava verkko voidaan rakentaa fyysisesti eri tavalla. Esimerkiksi IP (Internet Protocol), jota voidaan käyttää lähiverkkoa, puhelinlinjaa ja langatonta satelliittiyhteyttä pitkin. Kerroksen tehtävä on siis valita, mitä reittiä pitkin tieto lähetetään. Kuljetuskerros takaa jatkuvan yhteyden eli huolehtii vaihtoehtoisten reittien käyttämisestä vian takia, ilman tietoliikenneyhteyden katkaisua. Yhteysjaksokerros pitää huolta, että tiedonsiirto ei mene ”vikatilaan”, fyysisten yhteyksien katketessa sekä huolehtii tarvittaessa tiedonsalaamisesta. Esitystapakerros huolehtii tiedon esitysmuodosta eli päättää, missä muodossa teksti, kuva tai ääni esitetään, että vastaanottaja ymmärtää tiedon. Sovelluskerros toimii linkkinä aina kuhunkin ohjelmaan, joka tarvitsee tiedonsiirtoa. (7, Kappale 19)

Tiedon salaaminen on keskeinen osa tiedonsiirrossa, sillä luottamuksellistakin tietoa halutaan siirtää, välittää ja säilyttää. Tietokoneiden yleistymisen myötä vanhat salausmenetelmät ovat käyneet tehottomiksi, joten salausmenetelmiä kehitetään jatkuvasti. Konventionaaliset menetelmät perustuvat salaiseen algoritmiin tai salasanan käyttöön. Molemmissa tapauksissa ongelmana on miten salasana tai algoritmi välitetään viestin vastaanottajalle niin että se ei päädy sivullisille. Julkisen avaimen salausmenetelmässä viestin salaamiseen ja purkamiseen käytetään eri avaimia. Julkisen avaimen avulla viesti voidaan salata ja viestin purkamiseen tarvitaan salainen avain. (7, Kappale 19)

2.4 Tietokanta

Tietokanta on tietokokonaisuus, joka koostuu samankaltaisista, toisia sitovista tiedoista. Tavallisesti tietokanta on yhteneväinen tietorakenne tiedon ylläpitoa, hakua ja selaamista varten. Tietokonesovellusten yleistyessä myös tietokannoista on tullut osa arkea, lähestulkoon kaikki internetissä sijaitseva tieto on tietokannoissa, joista se koodiin sisällytettyjen tietokantakyselyiden avulla käydään hakemassa, päivittämässä tai poistamassa. (8, Tietokanta; 9, Tietokannat)

Relaatiomalli on tietokantamalleista yksinkertaisin, joten se on valloittanut tietokanta-arkkitehtuurin verrattuna verkkomalliseen tietokantaan, koska relaatiomallin käyttäminen ja muokkaaminen on huomattavasti helpompaa. Se muodostuu taulusta tai tauluista ja niiden yhteyksiä kutsutaan relaatioiksi. Taulussa on sarakkeita eli kenttiä ja rivejä eli tietueita. Kenttää määrittäessä tulee tietää, minkä tyyppistä tietoa siihen tallennetaan sekä tiedon pituus maksimissaan. Avainkenttä eli perusavain yksilöi tietueen ja yhteys kahden taulun väliin muodostetaan käyttämällä perusavainta toisen taulun kentäksi, jota kutsutaan nimellä viiteavain. Tietokantojen koot voivat vaihdella tarvittavien sovellusten mukaan, minimaalisista miljooniin tietueisiin. Viite-eheys eli taulun kentän viittaamisesta toisen taulun perusavaimeen tai toissijaiseen avaimeen antaa taululle tarvittavat relaatiot. Viiteavaimeksi kutsutaan kenttää, jolla viitataan toisen taulun perusavaimeen. Taulujen relaatio esitetään kuvassa 7. (8, Tietokanta; 9, Tietokannat)



Kuva 7. Kuvaus taulujen relaatioista. (8, Tietokanta)

Tunnetuimpia tietokannan hallintajärjestelmiä ovat MS SQL Server, MySQL, Oracle, PostgreSQL, Ingres, DB2, Informix, Solid ja Firebird. Uutena pilvipalvelutietokantana on tullut MariaDB, jonka kehittämisen on aloittanut MySQL:stä tuttu ydinryhmä. Tietokannanhallintajärjestelmiin lukeutuvat myös henkilökohtaiset tietokantaohjelmat Access ja Paradox. Henkilökohtaisia tietokantajärjestelmiä ei ole suunniteltu tallentamaan suurta määrää tietoa, joten ne eivät ole niin kattavia. SQL Serveristä on olemassa Expressversio, joka on ilmaiseksi ladattavissa oleva versio, eikä ole niin

kattava kuin maksulliset versiot. MySQL voidaan luokitella myös samaan kategoriaan kuin SQL Server Express Edition. (10, Yleistä)

SQL on IBM:n kehittämä relaatiotietokantojen käsittelyyn ja hallintaan kehitetty kieli, jota kaikki relaatiotietokannat ymmärtävät. SQL-kielellä kerrotaan mitä haetaan eikä miten haetaan. Sitä voidaan myös upottaa ohjelmointikieleen, joita ovat esimerkiksi Visual Basic, C#, Cobol tai Java. SQL-kielellä hallinnoidaan tietokantaa, sen käsittelyä ja ylläpitoa. SQL-kieli voidaan jakaa kahteen osaa, jotka ovat tiedonmäärittelykieli ja tiedonkäsittelykieli. Määrittelykielellä luodaan ja muokataan käyttäjäoikeuksia, tietokannan rakennetta ja proseduureja, kun käsittelykieli vastaa kyselyistä ja ylläpidosta. SQL-tietokantakielellä voidaan luoda ja muokata tietokantaa ja käyttöoikeuksia, hakea kannasta tietoa ja tehdä päivityksiä tietokantaan. Yleisimmät SQL-kyselyt on listattu taulukkoon 3. (10; 11)

Taulukko 3. SQL-kyselyitä.

SELECT * FROM taulu	Valitsee kaikki kentät taulusta
INSERT INTO taulu (kenttä1, kenttä2) VALUES ('arvo1', 'arvo2')	Lisää uuden rivin ja arvot tauluun
UPDATE taulu SET kenttä = 'arvo1' WHERE kenttä = 'arvo2'	Päivittää taulusta kentän arvo1:n arvo2:lla
DELETE FROM taulu WHERE kenttä = 'arvo1'	Poistaa kentän, jonka arvo 1 taulusta

3 Laadunhallinta

Laadunhallinnan kannalta on hyödyllistä tutustua laadunhallintamalleihin. Standardit käsittävät laadunhallinnan kannalta oleellisia osia, joita voidaan verrata omiin, asiakkaan tai alihankkijan prosesseihin. Laadunhallinta jaetaan kahteen osaan, joita ovat laadunohjaus ja laadunvarmistus. Laadunohjaus määritellään käyttämällä määriteltyjä toimintoja ja tekniikoita, joiden avulla pyritään täyttämään laatuvaatimukset. Laadunvarmistuksella pyritään saamaan sisäisten ja ulkoisten asiakkaiden, sekä viranomaisten luottamus laatutavoitteiden toteutumiseen. Laatujärjestelmän rakenne jakautuu osiin, joita ovat laatuun liittyvien toimintojen vastuut ja valtuudet, organisaatiorakenne, resurssit ja koko henkilöstö sekä operatiivinen menettely. (12, Laadunhallinta)

Projekteissa dokumentit koostuvat yleisesti, laatukäsikirjasta, projektikohtaisesta laatusuunnitelmasta ja testaus-, tarkastus-, ja valvontamenettelyiden tuloksista. Laatupolitiikka on kuvaus koko yrityksen kattavasta yleisluontoisesta kuvauksesta laatutavoitteille, kun taas projektikohtaisessa laatusuunnitelmassa laatupolitiikka muutetaan mitattaviksi laatutavoitteiksi. Laatujärjestelmien toimivuutta ja noudattamista on tarkastettava tietyin välein sisäisellä auditoinnilla. Korjaavia toimenpiteitä suoritetaan, kun työskentelymenetelmiin tarvitaan muutoksia. Korjaavat toimenpiteet tulee määritellä huolellisesti ennen niihin ryhtymistä. Tämänkaltaisia määrittelyyn liittyviä asioita ovat ongelman määrittely, vastuiden määrittäminen, toiminnon tärkeys, syiden määrittely, analyysi syiden määrittelystä, tapahtuman uusiutumisen estämismahdollisuudet, ratkaisu ongelmaan ja muutokset. (12, Laadunhallinta)

Tarkastusmenettelyissä päätarkoituksena etsitään virheitä dokumenteista ja ohjelmista eikä niiden tekijöistä. Niin kuin monessa muussakin asiassa, tekijä itse ei näe omia virheitään niin hyvin kuin toinen tarkastaja, joten tarkastuksen suorittajan kannattaa olla toinen henkilö kuin tekijä itse. Tarkastusmenettely motivoi työntekijää panostamaan huolellisemmin tehtävään, tietäen sen menevän toisille tarkastettavaksi. Tarkastusmenettelyn tavoitteita ovat virheiden löytyminen mahdollisimman aikaisin, varmistua siitä, että kaikilla on sama käsitys asioista, jakaa tietoa, todentaa, että työ on vaatimusten mukainen, antaa päätös työvaiheelle sekä tuottaa mitattavaa tietoa tuotteesta. Tarkastusmenettelyn eri muotoja ovat tarkastus, katselmus, läpikäynti ja hallinnollinen katselmus. (12, Laadunhallinta)

Tarkasteluvaiheet jakautuvat kolmeen osaan, joista ensimmäisenä on valmistelu. Valmistelu käsittää tekijöiden ja johtajien päätöksen tuloksen tarkastamisesta ja päätöksen tarkastuksen tavoitteista. Tarkastuksen tavoitteet kuvataan aloituskokouksessa, jossa jaetaan tarkastustoiminnan materiaali. Jokainen tarkastaja käy materiaalin läpi ja kirjaa havaitsemansa virheet ja epäselvyydet. Näin saadaan virheitä karsittua valmistelun aikana. Toisena tarkasteluvaiheena on tarkastustilaisuus, jossa käydään läpi tarkastettava dokumentti ja tehdään tarvittavia korjauksia. Kolmantena tarkasteluvaiheena on jälkihoito, jossa toimitetaan pöytäkirjan kopiot osallistujille. Tekijät korjaavat havaittuja virheitä ja käyvät korjaukset läpi valvojan kanssa. Lopuksi valvoja tarkistaa, että kaikki tarkastuksen vaiheet ja toiminnot on suoritettu. Tyypillisiä syitä tarkastuksen epäonnistumiseen on valmistelun puutteellisuus, liian iso tiimi, väärät henkilöt tai tarkastettava materiaali liian suuri. (12, Laadunhallinta)

4 Vaatimusmäärittely

Jokaisen testatun elektroniikkakortin tuloksena luodaan yksi XML-tiedosto, jossa on tietoja testin kulusta sekä asetus- ja tulostusarvoista. XML-tiedostot tallennetaan testerikoneen kovalevylle, joita syntyy päivässä tuhansia yhdelle tuotantotesterille. Tiedosto sisältää paljon ylimääräistä eli ei hyödynnettävissä olevaa informaatiota testistä. Testauksessa luoduista testiraporteista halutaan siirtää hyödylliset testaustulokset tietokantaan, joita ovat testin yleistiedot ja mittaustulokset. Tarvittavia tietoja ovat COMMON-elementissä olevat testin yleistiedot sekä CHK_***-elementeissä olevat mittaustulokset. Elementtien komentotiedot kuvassa 8.

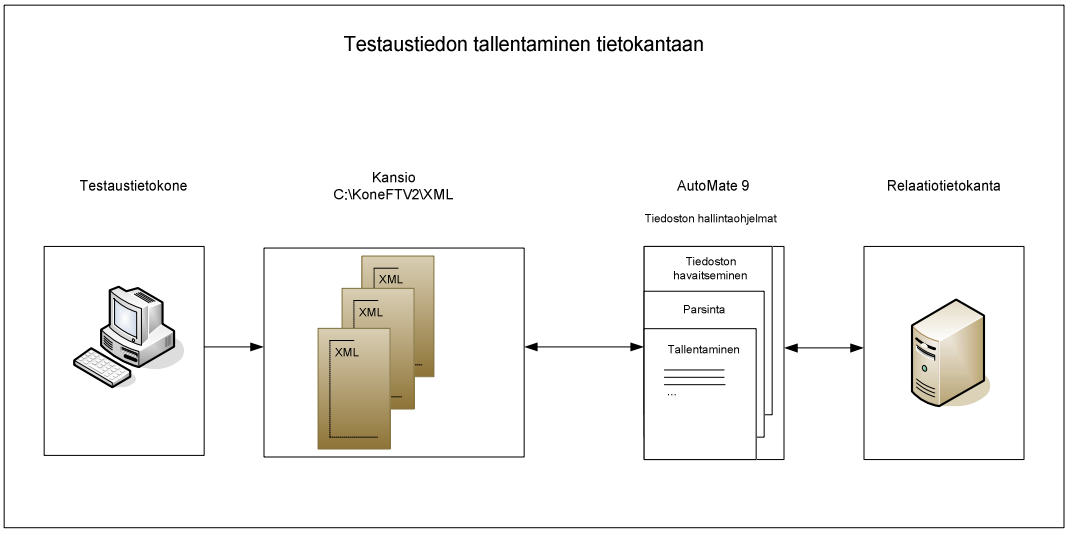
TestCommands				
COMMON	CHK_DLL_PROGRAM	CHK_SER_MEM	CHK_KDH_IO	RS232_Command_CHK
CHK_SER	CHK_DLL_VERSION_X-Y-Z	CHK_SER_SERIAL	CHK_KDH_VE	LIMIT_Command_CHK
CHK_DMM	CHK_DLL_SCALING	CHK_SER_LINK	CHK_KDH_MSW	
CHK_FILE	CHK_DLL_RSWITCH	CHK_SER_INPUT	CHK_KDH_MSR	
CHK_LON	CHK_DLL_STATUS	CHK_SER_CAN	CHK_KDH_SPIx	
CHK_DLL	CHK_DLL_FREQUENCY	CHK_SER_CPU	CHK_PAR_PROGRAM	
CHK_DLLX	CHK_DLL_ERROR	CHK_SER_BUSRD	CHK_PAR	
CHK_LON_SERVICEP	CHK_DLL_SPEED	CHK_SER_ADON	CHK_PS1	
CHK_DLL_SERVICEP	CHK_SER_FLASHSECT	CHK_SER_VA1_VAR1_ VAR2_VAR1	GLOB_Command_CHK	
CHK_DLL_SIDE	CHK_SER_COM	CHK_SER_SW	RS423_Command_CHK	

Kuva 8. Tarvittavien elementtien nimitiedot.

Toiminnot suoritetaan AutoMate 9 -ohjelmistolla, johon KONEen tuotekehitysyksiköllä on lisenssi, jota on aiemmin käytetty testereihin liittyvissä tuotekehitysprojekteissa.

Tietokannaksi valitaan tunnetuista tietokannoista mieluisin ja samalla tietokannan hallintajärjestelmä, kyseiseen käyttötarkoitukseen sopii myös ilmaisversiot. Tietokannan suunnittelu eli sen rakenne ja sisältö jätetään pois kirjallisesta esityksestä.

Kun testeri tuottaa uuden XML-tiedoston, ohjelma havaitsee ja suorittaa tarvittavat jatkotoimenpiteet, joissa parsitaan testin yleistiedot sekä mittaustulokset ja tallennetaan tietokantaan. Valmiin työn tulee olla asennettavissa testerikoneelle. Vaatimusmäärittelyn mukaisen järjestelmän kokonaiskuvaus kuvassa 9.



Kuva 9. Tiedon tallentamisen kokonaiskuvaus.

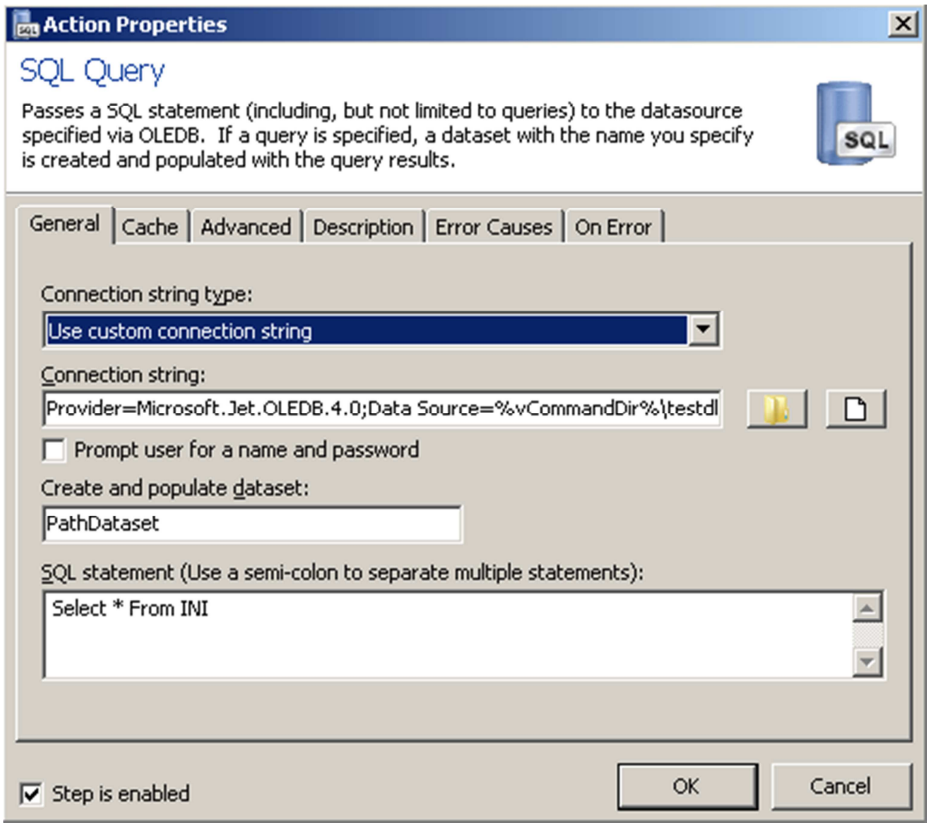
Työn ohella havaittavista ongelmakohdista tai huomioista raportoidaan työn tilaajalle ja tapauksen mukaan tehdään raportit huomioista.

Projekti määritellään valmiiksi kun ohjelma toimii täysin itsenäisesti ja tarvittava kirjallinen osio on suoritettu.

5 Toteutus

5.1 Tarvittavat ohjelmistot

Projektissa käytettävä AutoMate 9 -ohjelmisto, millä toteutetaan tarvittavat ohjelmat suunnitelmien mukaan, on Microsoft Windows ympäristöön tarkoitettu IT-automaatio-ohjelmisto. AutoMatella voidaan suorittaa Windows-käyttöjärjestelmän operoinnin automatisointeja rekistereistä lähtien aina kansioiden luontiin. Ohjelmointi tapahtuu valmiilla komponenteilla kuvassa 10 tai basic-ohjelmointia muistuttavalla ohjelmoinnilla. Valmiit komponentit ovat windows-ikkunoita, joihin määritellään parametrit. Ohjelmiston on kehittänyt networkautomation niminen yritys Los Angelesista. (13, Products; 14)



Kuva 10. AutoMaten SQL-kyselykomponentti.

Tietokantajärjestelmäksi valitaan MySQL-tietokanta, jota operoidaan phpMyAdmin tietokannanhallintajärjestelmällä. PHP:tä ja Apache HTTP Serveriä tarvitaan phpMyAdmin tueksi, joka toimii tietokannan hallintajärjestelmänä. Tietokanta ja

tietokannanhallintajärjestelmä sekä niihin liittyvät oheisohjelmat ovat ladattavissa ilmaiseksi kunkin järjestön internet-sivuilta.

5.2 Relaatiotietokanta

Relaatiotietokanta suunnitellaan aikaisempien tietokantaratkaisujen kanssa rakenteeltaan samanlaiseksi, jolloin tietoja voidaan jatkossa integroida pienemmällä vaivalla. Valmista tietokantaratkaisua verrataan kyseiseen sovellukseen sopivaksi ja mahdollisesti malliratkaisu voidaan ottaa käyttöön varmistamisen jälkeen. Varmennuksessa käydään tietokantarakenne läpi työn vastaavan kanssa ja todetaan tietokantamallin tila. Tarvittaessa luodaan ja suunnitellaan alusta alkaen uusi ja hyödynnettävissä oleva relaatiotietokanta.

MySQL-tietokannan ylösajon teknisistä ongelmista, sekä työn prioriteetista johtuen päädyttiin Microsoftin SQL Server 2005 Express Edition -ohjelmistoon Windows XP-käyttöjärjestelmässä. Tarvittava tietokanta ohjelmisto Microsoft SQL Server 2005 Express Edition sekä tietokannan hallintajärjestelmä SQL Server Management Studio. Serverin ja käyttöliittymän asentaminen onnistuivat ongelmitta ohjelmien ”velho-työkaluilla”.

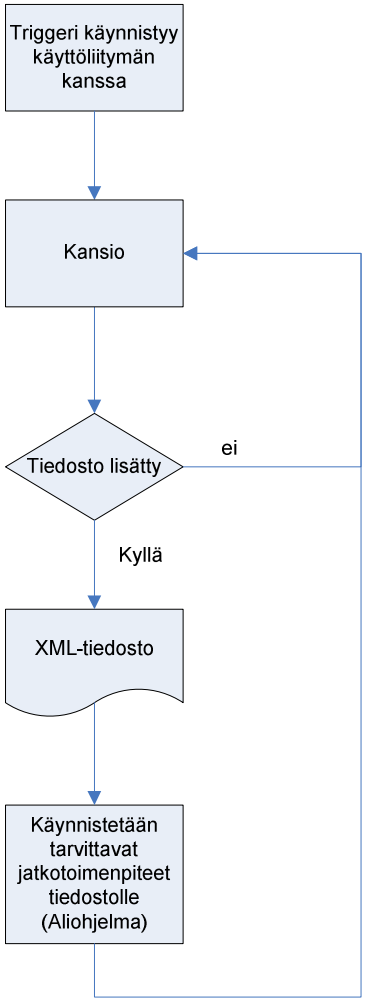
Tietokannan rakenne toteutettiin uudemman testerin tietokantarakennetta noudattaen, että tietokannoista saatiin yhtenevät. Tutkiessa tietokannan operoimista päätettiin käyttää hyväksi MS Access -tietokannan SQL Server -työkalua, jolla saadaan Access-tietokanta siirrettyä SQL Serverille kokonaisuudessaan. Kun todettiin MS-tuotteiden olevan yhteensopivia, käytettiin SQL Server työkalua ohjeiden mukaan. Tämä toiminto onnistui ja tietokanta siirrettiin SQL Serverille. Toiminto kopioi Access-tietokannan rakenteen, sekä taulukot kokonaisuudessaan ilman tallennettuja arvoja ja määriteltäessä arvojen kanssa.

Tietokannan loppuratkaisuksi tuli MS Access -tietokanta. Alun suunnitelmien mukaan käytettävä tietokanta olisi ollut MS SQL Server, mutta yhteysongelmien ilmetessä vikaa ei lähdetty syvällisemmin tutkimaan, koska tietokannan järjestelmä ei suoraan ollut työn kannalta ratkaiseva asia. Tietokanta saatiin tuotannossa olevalta testeriltä, joka käsitti määritysten mukaisen tietokantarakenteen. Tietokanta on mahdollista vaihtaa toiseen järjestelmään, ohjelmakoodin SQL-kyselyiden yhteyksien muutoksella. Tietokanta

käsittää tarvittavat tiedot, joita voidaan hakea eri SQL-kyselyiden avulla muihin ohjelmakoodeihin sisällytettynä. Näin ollen kyseistä tietokantaa ja sen sisältöä voidaan käyttää pohjana rakennettaessa käyttöliittymää, jonka avulla havainnollistetaan tietokannan aineisto visuaalisempaan muotoon.

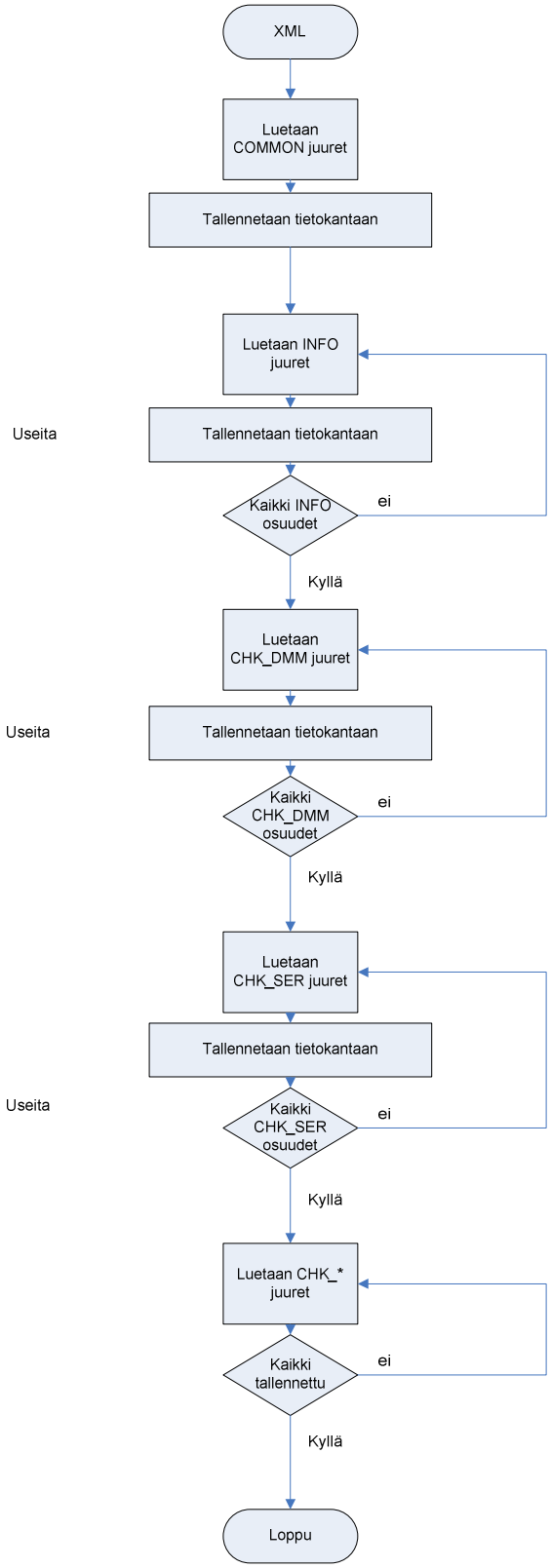
5.3 Ohjelmisto

Ohjelmisto-osuus voidaan jakaa karkeasti kolmeen ryhmään, joita ovat kansion sisällön muutosten havaitseminen kuva 11, XML-tiedoston parsiminen ja määriteltyjen parametrien tallentaminen tietokantaan. Kokonaisuus toteutetaan ohjelmoimalla sovelluksen eri osia, jotka integroidaan yhteen työn edetessä ja ohjelmistotestausta suoritetaan osien toteutuksen yhteydessä. Tämän kaltaista toteutustapaa käytetään läpi työn. XML-tiedostojen tallennuskansiota tulee tarkkailla uusien tiedostojen havaitsemiseksi, vuokaaviokuva havainnollistaa tapahtuman ehdot. Jos havaitaan uusi kohde, haetaan tiedoston nimi ja tallennuksen päivämäärä muuttujiin sekä suoritetaan tarvittavat jatkotoimenpiteet tiedostolle.



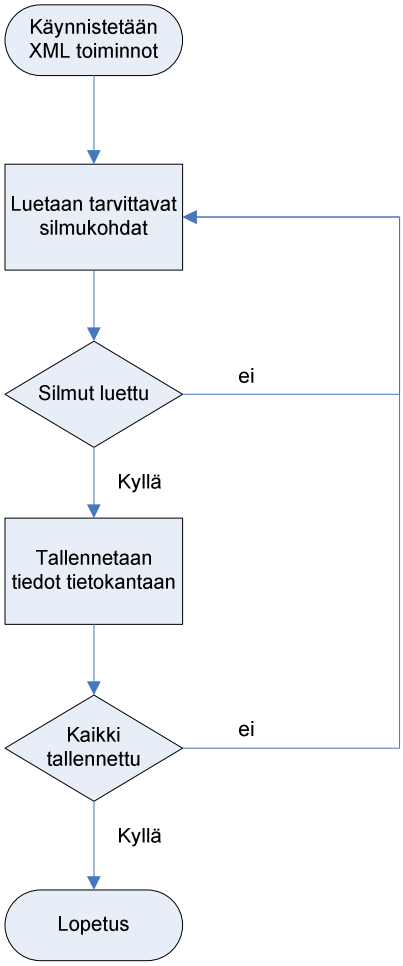
Kuva 11. Tiedoston tarkkailu.

Tiedoston havaitsemisen jälkeen aloitetaan XML-tiedoston parsiminen. XML-tiedoston parsinnan käyttöön tarvitaan perustietoja Xpath, DTD ja XSLT standardeista. Toiminnot aloitetaan lukemalla ensin tiedostosta yleistiedot COMMON osiosta muuttujaan ja tallennetaan ne tietokantaan. Seuraavaksi luetaan INFO tiedot ja suoritetaan niille tarvittavat tallennukset. Viimeisenä on itse mittauksien tulostiedot, jotka alkavat merkillä CHK_XXX ja käydään läpi kaikki tarvittavat elementit tiedostosta ja tallennetaan halutut tiedot tietokantaan. Kuvassa 12 vuokaavio tiedonparsimisen ehdoista.



Kuva 12. Tiedonparsiminen

Tietokantaan tallentaminen tapahtuu, kun elementin kaikki silmut on luettu. Kaikkien tietojen tallentamisen jälkeen lopetetaan ohjelmat. Kuva 13 havainnollistaa vuokaavion avulla tallentamisen ehdot.



Kuva 13. Tallentaminen tietokantaan.

Ohjelmointia tuki ohjelmisto-osuuden alkaessa hyvin suunniteltu ohjelmistosuunnittelu-osuus, jonka pohjalta rakennettiin ohjelma. Ohjelmisto jaettiin suunnittelu vaiheessa kolmeen osaan, joiden perusteella toteutusta rakennettiin.

5.3.1 Triggeri

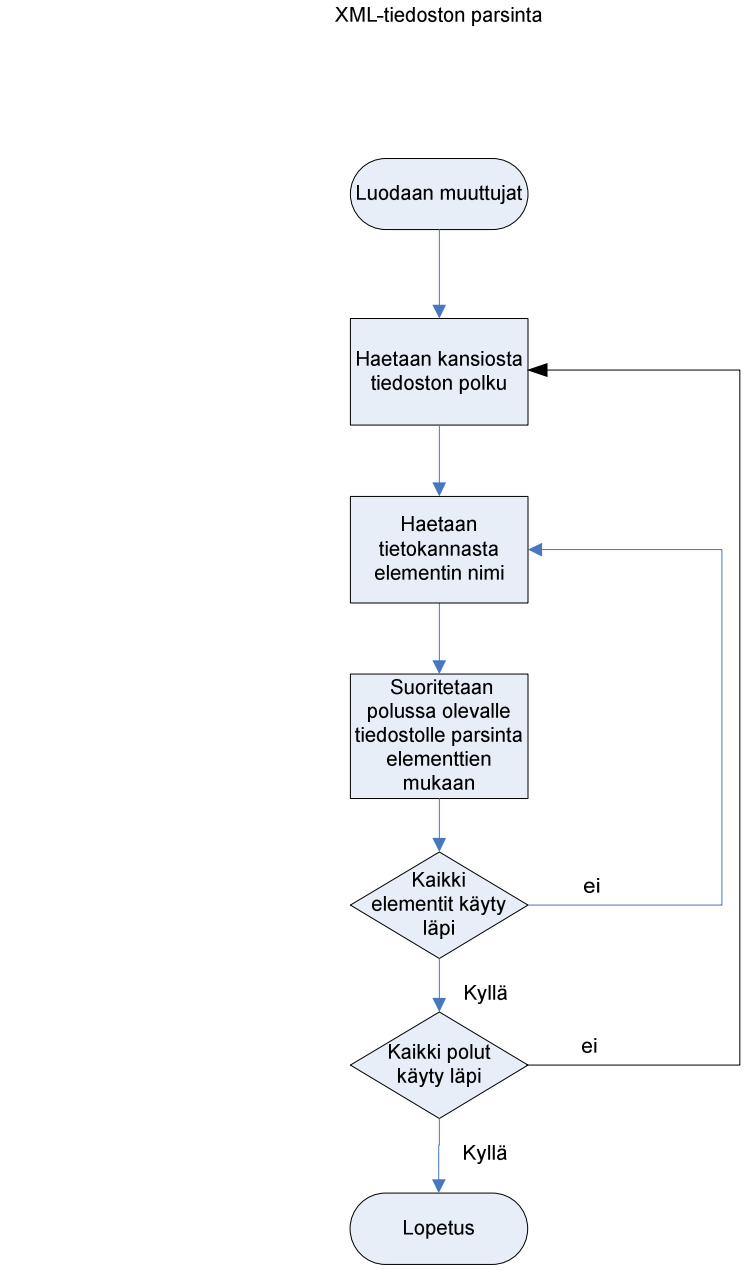
AutoMate -ohjelmistossa muutoksien havainnointi voidaan suorittaa erilaisilla triggeritoiminnoilla, joilla pystytään käynnistämään tehtäviä. Tehtävien aloitus mahdollisuuksia ovat Windowsin login muutokset, kansioiden muutokset, hiiren tai näppäimistön liikkeet, suorituskyvyn muutokset, prosessien muutokset, päivämäärän ja ajan mukaan, palveluiden muutokset, SNMP (Simple Network Management Protocol) -yhteyden mukaan, käyttöjärjestelmään kirjautumisen yhteydessä, määritelty ikkuna aukaistaan, sekä WMI Service -muutokset. Tämänkaltaisten muutosten avulla voidaan aloittaa tehtävien suorittaminen.

Tehtävän aloitus tässä työssä suoritettiin aika ja päivämäärä triggerillä, joka aloittaa tehtävän suorittamisen kerran päivässä määriteltynä ajankohtana. Tähän tulokseen päädyttiin työnohjaajan kanssa, ratkaisun ollessa käytännöllinen ja prosessiin sopiva.

5.3.2 Tietojen parsiminen

Tietojen parsimisen mahdollisuuksia ovat AutoMaten XML-operaatiot, joilla voidaan suorittaa tiedostoille kattavia toimenpiteitä. XML-istunnon aloitus osiossa tarkastetaan, onko tiedosto formaattien mukainen ja havaittaessa poikkeavuuksia tiedostossa suoritetaan komponenttiin määritellyt vikatoiminnot. Jos tiedosto läpäisee istunnon eli toiminto vahvistaa dokumentin formaatin olevan kunnossa, voidaan tiedostolle suorittaa XML-toimintoja. Vahvistamisen jälkeen elementtien arvoja voidaan muokata ja lukea. XML-istunto lopetetaan, kun määritellyt XML-toiminnot on suoritettu.

Tämän osion ratkaisu aloitettiin vahvistamalla dokumentti, jonka jälkeen tiedostolle suoritettiin elementtien arvojen lukeminen. Hylätylle dokumenttiformaatille suoritettiin alasajo, jossa kirjoitetaan vian tiedot vikatietologiin. Tarvittavien elementtien nimet haettiin suorittamalla SQL-kysely tietokannasta ja arvot tallennettiin muuttujaan. Elementin arvo luettiin muuttujaan XML-formaatin polun perusteella. Elementtien etsimisessä käytettiin apuna XML-kieltä eli XPath-rakennetta, joilla saatiin suoritettua tarvittava elementtien haku tiedostosta. Perus XPath-komentoja hyödyntämällä saatiin tarvittava ratkaisu toteutettua. Kuvassa 14 tiedoston parsinnan ehdot toteutuksen mukaisesti.



Kuva 14. Toteutuksen mukainen tiedoston parsinta.

5.3.3 Parametrien tallentaminen tietokantaan

AutoaMate mahdollistaa tietokantakyselyiden suorittamisen SQL-tietokantoihin omilla komponenteilla. Kyselyitä voidaan suorittaa SQL Query -komponentilla. Tietokannasta lukeminen voidaan toteuttaa ilman erillistä yhteydenavauskomponenttia ja kirjoitettaessa tietokantaan tulee yhteys avata open komponentilla. Parametrit tallennettiin tietokantaan käyttämällä open-, close- ja query-komponentteja. Tallentamisessa käytettiin yhtä muuttujaa, johon tallennettiin elementin arvo ja suoritettiin muuttujan tallennus SQL Query -komponentilla. SQL-toiminnon havaitessa virhetilanteen, suoritettiin toiminnon alasajo kirjoittamalla talteen SQL-toiminnossa havaittu vikatieto SQL vikatietologiin. Työssä käytetyt SQL-komennot on listattu taulukkoon 4.

Taulukko 4. Työssä käytetyt SQL-komennot.

Komento	**Määritys**
SELECT * FROM ”taulun nimi”	Kaikki kentät taulusta
INSERT INTO (”Kenttä”) VALUES (’ ”Arvo” ’)	Lisätään uusia arvoja määriteltyyn kenttään
SELECT TOP 1 ”Kenttä” FROM ”Taulu” ORDER BY ”Kenttä” DESC	Haetaan uusin arvo määritellystä taulun kentästä
UPDATE ”Taulu” SET ”Kenttä” = ’ ”Arvo1” ’ WHERE ”Kenttä” = ”Arvo2”	Päivitetään olemassa olevalle riville kentän tietoja

Lopullisen ohjelman pääkohdat ovat muuttujien luonti, INI-tietokannasta asetuspolkujen haku, kansioiden olemassa olo, komentojen haku ja vertailu XML-tiedostoon, parametrien tallentaminen ja vikatilanteen alasajo. Ohjelman rakenne havainnollistetaan kuvassa 15.

Aloitus	
Muuttujat	
INI tietokanta	
Kansioiden luonti	Vikatilanteen alasajo
Komentojen haku ja vertailu XML tiedostoon	
Parametrien tallentaminen	
Lopetus	

Kuva 15. Ohjelman rakenne.

5.4 Ohjelmistotestaus

Työn testausvaiheessa käytettiin ohjelmistotuotannon laadunhallinnan elinkaarimallia hyväksi, joka koostuu kuvan 16. mukaisesti.

Kuva 16. Ohjelmistotuotannon laadunhallinnan elinkaarimalli. (1, s. 26.)

Ohjelmistotestausta suoritettiin toteutuksen ohella aina yhden osion valmistuessa ja uusia osia integroitaessa. Suurimpia osia, joista integroinnit koostuivat, ovat tiedoston istunto, tiedoston polun haku, komentojen haku tietokannasta, tiedoston elementtien etsiminen, vikatilanteen alasajo ja istunnon lopetus.

XML-tiedoston istunnon aloituksessa havaittiin testauksen ohella tulevan virhe, jonka takia istuntoa ei voitu aloittaa. Virhettä selvitettäessä etsittiin tiedostosta kohta, johon vikatieto viittasi. Tuloksena havaittiin rivillä oleva merkki ja se että tiedosto poikkeaa standardista eikä istuntoa voida aloittaa ilman tiedostoon tehtäviä korjauksia.

Tämän seurauksena testausta jatkettiin etsimällä standardin mukainen tiedosto, jolla istunto voidaan aloittaa. Standardin mukaisella istunnolla testaus onnistui ja XML:ään kohdistuvat toimenpiteet aloitettiin. Komentojen haussa testauksena oli SQL-kysely ja kyselyn tuloksen tallentaminen muuttujaan. Osa testattiin omana toimintanaan ja tehtävälle suoritettiin integrointitestaukset, jotka onnistuivat ilman ongelmia. Seuraavana testauksessa oli elementtien hakutoiminnon ja arvon asettaminen haluttuun muuttujaan. Testaus suoritettiin onnistuneesti ensimmäisellä testauskerralla. Yleistieto-osion ollessa laajempi, sitä testattiin useampia kertoja erikseen ja osaan tehtiin tarvittavia korjauksia. Laajemman testauksen tuloksena saatiin hyvä kokonaiskuva osiosta, jota lähdettiin liittämään varsinaiseen parsintaohjelmaan ja suorittamaan sille tarvittavat testit. Mittaustulos-osion testauksessa tuli esille tarvittavia SQL-kyselyihin liittyneitä havaintoja, jotka auttoivat parantamaan ohjelman laatua kokonaisuudessaan. Tämän seurauksena saatiin suurimmat huomiot havaittua ja osio oli valmiina liitettäväksi. Kun liitettiin yleistulos ja mittaustulos valinnat parsintaohjelmaan, testattiin tallentamisen onnistumista. Testauksessa havaittiin ongelmat SQL-komponenttien vikatilanteen ilmetessä ja niille suoritettiin tarvittavat jatkotoimenpiteet. Istunnon lopetuksen toiminta testattiin ensimmäisellä istunto testauksella, joka sisälsi lopetuksen ja aloituksen.

Tähän mennessä saatiin suurin osa kokonaisratkaisusta yhteen ohjelmaan, jonka jälkeen testattiin ohjelma, joka hakee halutusta kansiosta tiedoston ja hakee tiedostosta tarvittavat tiedot, tallentaa tiedot muuttujaan ja muuttujan arvo tallennetaan tietokantaan. Kun kaikki halutut tiedot on tallennettu, lopetetaan kyseisen tiedoston istunto ja aloitetaan kansion seuraavan tiedoston parsiminen ja suoritetaan samat toimenpiteet kaikille kansiossa oleville tiedostoille.

Laajempi testaus suoritettiin kansiolle, joka sisälsi noin 700 tiedostoa. Testauksen tuloksena saatiin kaksi eri kansiota, johon tiedosto siirrettiin testaamisen jälkeen riippuen toteutuksen onnistumisesta. Fail_Xml_Session-kansioon siirrettiin noin 520 tiedostoa, koska ne eivät läpäisseet tiedostoille asetettua formaatin vahvistusta, joten niille ei voitu aloittaa istuntoa. Pass_Xml_Session kansioon siirrettiin noin 180 tiedostoa, joista halutut tulokset saatiin tallennettua tietokantaan. Tämän testauksen avulla saatiin selville, minkä kortin tiedostot eivät noudata standardia. Error_log tiedostoa tutkiessa huomattiin viallisten tiedostojen sisältävän & tai > merkit tulosalueella, jotka havaitaan taulukossa 5.

Taulukko 5. Standardista poikkeavia elementtejä.

Tiedoston elementti	Virhe merkki
<Comment>ARE "+24V" "+5V" "+12V" AND "-12V" LEDS ON? (ALSO 77U & 77N)</Comment>	&
<Comment>Check to be <10ohm</Comment>	<
<Comment>4.0.- ALTERNATE LON CONNECTOR & DOOR SIDE</Comment>	&
<Comment>4.4 EPROM VERSION TEST (8 = H01 & H02; 83 = H03)</Comment>	&
<Comment>4.6 *** Verify LEDs L18 & L19 are ON/OFF when PS1 is pressed/released</Comment>	&

6 Yhteenveto

6.1 Työn tarkastelu

Työ aloitettiin käymällä läpi opinnäytetyöaiheita eri osa-alueilta, ja testeripuolelta nousi mielenkiintoinen projekti esille. Aloituspalaverissa hahmoteltiin, mitä työ pitäisi sisällään ja mitä siihen voisi kuulua. Tarkoituksena oli rajata asioita ja selventää, minkälaisia seikkoja tulee ottaa huomioon. Tiedon hankinnan ja vertailun jälkeen taustat alkoivat hahmottua, ja seuraavassa speksauspalaverissa selvitettiin työn suuntaa. Työ esiteltiin koulun edustajalle ja siihen saatiin erilaista näkökantaa. Sisällysluettelo annettiin kommentoitavaksi ja kommenteista otettiin hyöty työn tekemiseen. Projektiaikataululla selkeytettiin sitä mihin projektia ollaan viemässä ja siitä seurattiin askeleittain työn kulkua. Sisällysluettelo ja aiheet elivät työn ohella, jolloin niistä muodostui relevantteja.

Projektissa käytettiin laadun varmistamiseksi CMMI-ohjelmistotuotannon laadunhallintamenetelmää, jonka avulla ohjelmistoprojektin suorittamiseen liittyvät asiat ratkaistiin järjestelmällisesti. Käytännössä ohjelmistotyössä suoritetaan vaatimusten kartoitus, jota analysoidaan. Seuraavana osiona suunnitellaan arkkitehtuuri ohjelmistolle. Komponenttisuunnittelun jälkeen suoritetaan koodaus ja yksikkötestaus, jonka jälkeen komponentit testataan. Sen jälkeen suoritetaan integrointitestaukset. Viimeisenä testauksena ovat järjestelmätestaus, jota verrataan vaatimusten analyysiin ja hyväksymistestaus, joka suoritetaan valmiille kokonaisuudelle ja verrataan alussa määriteltyihin vaatimuksiin. Ohjelmistoprojektissa todettiin testauksen olevan avainasemassa toteutusta suoritettaessa. Testausta tulee suorittaa monelta eri näkökannalta, jolloin havaitaan erilaisia ohjelman suoritukseen liittyviä ongelmia.

Tietokannan suunnitteluosuus jätettiin työn kirjallisesta osiosta pois ja tietokantana käytettiin valmista tietokantarakennetta, joka helpottaa tulevia tietokantaintegrointeja. Ohjelmisto suunniteltiin pääalueiden ympärille, joiden avulla suunniteltiin komponentit. Ohjelmistotestaus kulki yhdessä ohjelmiston toteutuksen kanssa, eli projekti rakennettiin osista, joita testattiin erikseen ja integrointien jälkeen, jolloin havaittiin ohjelmassa olevat puutteet ja suoritettiin tarvittavia toimenpiteitä.

Ohjelman rakentaminen vaihe vaiheelta kulki testauksen kanssa yhdessä, niin kuin ohjelmistoprojektin laadunhallinta elinkaarimalli havainnollisti. Ohjelmistokoodauksen

edetessä ohjelmiston ideologia selkeni, jonka seurauksena koodausosio nopeutui huomattavasti alun vaiheista. Rakennettaessa ohjelmaa suunnitelmien pohjalta, tulee siihen väistämättä eteen poikkeavuuksia, jotka havaitaan ohjelman koodausvaiheessa tai vasta testausvaiheessa. Tällaisessa tapauksessa huomataan, kuinka suunnitelmia hiotaan tapauskohtaisesti toteutuksessa. Tässäkin työssä verrattaessa suunniteltua ja lopullista ohjelmistoa, havaitaan lopullisen version poikkeavan suunnitellusta.

Suurena apuna toteutusta oli kattava help-tietopaketti, jonka avulla saatiin selville AutoMaten komponenttien toimintaa ja työkalun käsittelyä. Networkautomationin verkkosivuilta löytyi myös valmiita ohjelmaesimerkkejä, jotka havainnollistavat ohjelmiston automatisoinnin hyötyjä. Verkkosivut sisälsivät valmiita perustoimintojen koodin pätkiä, joilla voi tarkistaa ohjelman operointityökalun toimivuutta käytännössä. Yrityksen verkkosivujen blogi-osiossa oli kattava tietokanta aiemmin esitetyistä kysymyksistä ja niiden vastauksista, joka hyödyntää ongelmatilanteiden ratkaisuja.

Ohjelma on Windows-käyttöjärjestelmälle tarkoitettu automatisointityökalu, mutta siinä on silti paljon yhteistä automaation ohjausjärjestelmien ohjelmoinnin kanssa. AutoMate on kätevä ja hyödyllinen automatisointityökalu käyttöjärjestelmäympäristöön. Sen yksinkertainen komponenttivalikoima ja kattava help-valikko tarjoaa hyvät lähtökohdat automatisoinnin suorittamiselle. Koodaaminen ei vaadi syvällistä koodausideologiaan perehtymistä. Ohjelmakoodin kirjoittamisessa huomattiin tietoikkunakomponentin käytön hyödyt. Ohjelmaan sijoitettaessa tietoikkuna, johon määritellään, mitä ikkuna tässä ohjelman askeleessa käyttäjälle näyttää. Esimerkiksi muuttujien jälkeen voidaan sijoittaa tietoikkuna, johon lisätään teksti, muuttujat luotu. Suurissa silmukkamäärissä on erittäin hyödyllistä käyttää tietoikkunakomponenttia, jolloin voidaan pysyä ohjelman suorituksen askelten mukana testausvaiheessa. Toteutuksen viimeiseen versioon päädyttiin hyödyntämällä suunniteltua ja rakentamalla versio, joka kattaa vaatimusmäärittellyssä esiin tulleet asiat.

Komponenttien rakentaminen järjestelmällisesti ja huolellisesti, mahdollistaa työn suorituksen järjestelmällisyyden. Jokainen komponentti on hyvä tallentaa erikseen talteen, jotta sitä voidaan hyödyntää mahdollisesti tulevissa ohjelman osioissa. Kun komponentit on tehty huolellisesti ja tallennettu, on ohjelmakomponentin sijoittaminen ja muokkaaminen toiseen osioon huomattavasti helpompaa. Jatkokehityksen kannaltakin voidaan tutustua ensin ohjelman eri komponentteihin ja niiden kopioihin tehdä tarvittavia muutoksia ja lopuksi vasta muutoksien tekeminen kokonaisratkaisuun.

Ohjelmaan tehtävien muutoksien ollessa pieniä, muutosten tekeminen kokonaisratkaisuun on järkevää, mutta tulee myös ottaa huomioon, voisiko muutetuista komponenteista rakentaa oman yhtenevän rakenteen, jolloin säästyttäisiin valmiin ohjelman muutosten ja integrointien ongelmilta. Samaa ideologiaa kannattaa käyttää väliversioissa. Tietyin välein kannattaa tallentaa väliversio ohjelmasta ja tehdä siihen kattavan tietopaketin mitä kyseinen väliversio sisältää. Näin ollen väliversioistakin voi saada suurimman osan kehitysprojektin ohjelmasta ja päästä ohjelman ideologiaan kiinni.

Ohjelmaa tehdessä olisi tärkeää tehdä riittävät kommentit koodista, jolloin toisen ohjelmiston kehittäjän on huomattavasti helpompi saada ohjelman rakenne ja osat selkeiksi. Jos tarvitsee jotain tallennettua komponenttia ja huomaa sen olevan hyödyllinen, niin riittävällä kommentoinnilla muutoksien tekeminen onnistuu, ja näin ollen muutoksien yhteydessä tulee myös ottaa huomioon muutoksien kommentointi.

Ohjelmistoprojektin edetessä tulee ottaa huomioon alkuasetusten konfigurointi. Jo pienessäkin ohjelmassa konfiguraatiotiedoston olemassa olo helpottaa konfigurointi muutoksien tekemistä. Tällaisia asioita ovat esimerkiksi tarvittavat tiedostopolut. Jos ohjelma ei sisällä niin sanottua ini-tiedostoa ja halutaan muuttaa jonkun tiedoston polkua, joudutaan muutos tekemään ohjelman sisälle. Kun taas ini-tiedoston ollessa käytössä, tarvittavat tiedostopolkujen muutokset voidaan tehdä suoraan ini-tiedostoon, ilman ohjelman avaamista ja muuttamista itse koodiin.

Koodirivien kasvaessa havaittiin, kuinka tärkeää on muuttujien nimeäminen. Jos muuttujat nimettäisiin ainoastaan niitä kuvaavilla nimillä, muuttujia olisi vaikea erottaa koodia kirjoitettaessa. Muuttujien nimeämiskäytännöt vaihtelevat tekijän mieltymysten mukaan, mutta lähtökohtaisesti muuttujaan kannattaa merkitä *v*, joka indikoi sanaa *variable*, sekä muuttujan tyyppi. Pieni v-kirjain kannattaa merkitä ensimmäiseksi kirjaimeksi muuttujaan, jolloin koodista havaitaan heti, mitkä nimet ovat muuttujia ja seuraavaksi tyyppi ja muuttujan määrittävä nimi. Tällaista rakennetta käytettäessä suuren ohjelmakoodin havainnointi helpottuu huomattavasti.

AutoMaten käyttöliittymä on käyttäjäystävällinen koostuen selkeistä pikavalintapainikkeista käyttöliittymän ylälaidassa, sekä sivulla olevasta komponenttikirjastosta, joka sisältää kaikki käytettävissä olevat komponentit. Komponentin määrittelevien tietojen löytyminen onnistuu F1-näppäintä painettaessa

komponentin ollessa valittuna. Tämä toiminto löytyy monesta eri ohjelmistosta, mutta kyseisen tapauksen informaatiosivut ovat johdonmukaiset ja kattavat. Rakennettua ohjelmaa tai komponenttia voidaan testata testiajolla ja käyttöliittymän alaosaan tulee tietoja testin kulusta ja mahdollisissa virhetilanteissa sinne indikoidaan vikakoodit ja sen selostus. Projekteja hallitaan Task Administrator -operaattorityökalulla, jonka avulla voidaan tutkia projektien tietoja ja päästä itse ohjelmointiosioon.

Lopputuloksena saatiin ohjelmisto, joka käy läpi halutusta kansiosta XML-tiedostot, suorittaa tiedostoille tietojen parsimisen, tallentaa yleistiedot ja mittaustiedot tietokantaan, lisää polkuun halutut kansiot, vikatilanteen osalta kirjoittaa vikatiedot error_log ja SQL_error_list -tekstitiedostoihin sekä lopettaa tehtävän, kun kaikki tiedostot on käyty läpi kyseisestä kansiosta. Työtä verrattaessa vaatimusmäärittelyn mukaiseen ohjeistukseen, voidaan todeta työn onnistuneen vaatimusten mukaisesti. Ohjelmistokokonaisuus toimii täysin automaattisesti, kun pakettikokonaisuus asetetaan oikeaan polkuun testerikoneelle. Ohjelma sisältää hyväksi havaitun ini-tietokannan, jossa määritellään tiedostopolut, sekä komentotietokannan, josta haetaan testiin tarvittavat komennot. Tiedostot varmistetaan tarkistamalla tiedoston luontiaika, jonka ollessa tarpeeksi suuri, voidaan tiedostolle määritellyt toiminnot aloittaa. Ohjelman tullessa virhetilaan, istunnolle suoritetaan tarvittava alasajo ja kirjoitetaan tekstitiedostoon havaitut vikatiedot. Vikatiedoista voidaan tarkistaa tiedoston nimi ja minkälaisia ongelmia havaittiin testin aikana.

Käytettäessä saman valmistajan tuotteita saatiin työ suoritettua ilman yhteensopivuusongelmia ja eri asetusten muuttamista skripteistä. Microsoftin konfigurointityökalua on johdonmukaista käyttää ja tarvittava konsulttiapu löytyy kätevästi Microsoftin internet-sivuilta.

Lähtökohtana toteutuksen onnistumiselle voidaan pitää työn prioriteetin tutkimista työn jokaisessa vaiheessa, näin voidaan varmistua työn oikeasta suunnasta. Opinnäytetyö kannattaa rajata mahdollisimman tarkasti ja heti alussa miettiä, mikä on työn pääaihe, jonka ympärille lähdetään rakentamaan kokonaisuutta. Kokonaisuuden havainnoinnissa suurena apuna toimii sisällysluettelon laadinta, josta saadaan selkeä kuva työn vaiheista. Projektin aikataulutus kannattaa laatia sisällyksen pohjalta ja suunnitella valmiiksi päivämääriä mahdollisille välietapeille. Yhtenevä ja johdonmukainen kokonaisuus antaa selkeän kuvan siitä, mitä on tehty, miten on edetty sekä minkälaisia ongelmakohtia työn edetessä on ilmennyt.

Työn edetessä havaittiin tärkeitä ongelmakohtia liittyen XML-tiedostojen formaattiin. Havainnoista laadittiin dokumentti, joka auttaa korjaavien toimenpiteiden suorittamista dokumentin luontiprosessiin. SQL-kyselyiden formaatissa havaittiin samankaltaisia ongelmakohtia, jotka estivät tallennuksien suorittamista. Vaadittavat korjaavat toimenpiteet liittyvät merkkeihin, joita ei saisi sisältyä XML-rakenteen arvokenttään.

6.2 Jatkokehitys

Tämän työn jatkokehitysmahdollisuudet liittyvät ohjelman suorittamisen tehokkuuteen. Siihen kuinka voidaan tehostaa ohjelman suoritusta eli minkälaisilla ratkaisuilla kokonais-suoritusaika saataisiin mahdollisimman lyhyeksi, pitämällä toiminnot vaatimusten mukaisina. Ohjelman suorittamisen kannalta ratkaisevia osia ovat XML-kyselyiden suorittaminen, SQL-kyselyiden suorittaminen sekä muuttujien määrän optimointi. SQL suorittamista voisi tehostaa tutkimalla optimaalisin tapa suorittaa tallennus tietokantaan ja selvittämällä tehokkain SQL-kyselyiden suorituskomponentti.

Esimerkkinä nyt on suoritettu yhtä muuttujaa tallentava ohjelma, joka tallentaa samaan muuttujaan elementin arvon ja tallentaa arvon tietokantaan, sen jälkeen tallentaa seuraavan halutun elementin arvon muuttujaan ja suorittaa SQL-toiminnon uudestaan, jokaiselle elementin arvolle erikseen. Jatkossa voisi paneutua siihen, mitä eri ratkaisuja komentojen suorittamiseen löytyy ja mikä kävisi juuri tämänkaltaiseen sovellukseen.

XML-tiedostolle voisi etsiä ratkaisua eri näkökulmasta, ilman yksittäisten tietojen parsimista esimerkiksi XML:n ja SQL:n mahdollisia muita integrointimenetelmiä.

Voisi edelleen tutkia ja mitata ratkaisua, johon lisätään muuttujat kaikille halutuille elementeille ja tallennetaan kaikki yleistiedon muuttujat yhdellä SQL-komennolla ja mittaustiedon muuttujat toisella. Tämä toiminto on suurelle datamäärälle parempi ratkaisu, mutta pitää pohtia, onko tämänkaltainen ratkaisu tehokkaampi kyseiselle ohjelmalle.

Ohjelmaan voisi suunnitella .INI-tiedoston, millä suoritettaisiin tarvittava konfigurointi, nyt sisältävän tietokanta konfiguroinnin sijaan. Olisiko se käytännöllisempi ja parempi ratkaisu?

Ohjelma voisi suorittaa tiedotuksen järjestelmän hallitsijalle sähköpostilla, lähettämällä error_log tiedostot ja yleistiedot määritellyin väliajoin.

Jos edellä mainittuja muutoksia tehtäisiin, miten ne vaikuttaisivat vikatilanteiden ilmenemiseen ja olisivatko vikatilanneongelmat erilaisia.

Lisäksi voisi vertailla eri ratkaisu mahdollisuuksia AutoMaten tilalle ja haastaa kyseisen ohjelman käyttöä tämänkaltaiseen sovellukseen, sekä voisi tutkia pilvipalveluvaihtoehtoa nykyiseen ratkaisuun, esimerkiksi minkälaiset hyödyt ja haitat pilvipalvelun käytöstä olisi.

Projektin jatkoksi voisi suunnitella tietokannalle visuaalisen käyttöliittymän, josta voidaan kätevästi tutkia tietokannan sisältöä ja saada hyödyllisiä raportteja analysointia varten.

Lähteet

1 Saari, Juha. 2013. Laadunhallintajärjestelmät. Luentosarja 7. Metropolia Ammattikorkeakoulu. Luettu 1.10.2013

2 SFS ry. 2011. Johdanto laadunhallintaan ja ISO 9000 –standardeihin. Kalvosarja oppilaitoksille.

3 KONE Oyj. Yrityksen verkkosivut. <www.kone.com>. Luettu 24.9.2013

4 Wikipedia. Verkkojulkaisu <http://fi.wikipedia.org/wiki/Bin%C3%A4%C3%A4ritiedosto>. Luettu 26.9.2013

5 W3C. XML. Verkkojulkaisu. <http://www.w3schools.com/xml/>. Luettu 27.9.2013.

6 2kmediat. XML. Verkkojulkaisu. <http://www.2kmediat.com/xml/>. Luettu 27.9.2013.

7 Tampereen yliopisto. Tiedonsiirto. Verkkojulkaisu. <http://www.cs.tut.fi/etaopetus/titepk/luku19/peruskasitteet.html>. Luettu 1.10.2013.

8 Sarja, Jari. Tietokanta. Verkkojulkaisu. <http://verkkopedagogi.net/vanhat/fi/sisalto/materiaalit/access2003/luku021c5a.html?C:D=419700&selres=419700>. Luettu 8.10.2013.

9 Jyväskylän yliopiston It-tiedokunta ja avoin yliopisto. Tietokanta. Verkkojulkaisu. <http://appro.mit.jyu.fi/doc/tiedonhallinta/tietokannat/index2.html¨>. Luettu 8.10.2013.

10 Hovi, Ari. 2012. SQL-opas. Luettu 1.10.2013.

11 Wikipedia. Verkkojulkaisu. <http://fi.wikipedia.org/wiki/SQL>. Luettu 8.10.2013.

12 Itä-Suomen Yliopisto. Laadunhallinta. Verkkojulkaisu. <http://cs.joensuu.fi/tSoft/laadunhallinta.htm>. Luettu 1.10.2013

13 AutoMate 9. Verkkojulkaisu. <www.networkautomation.com>. Luettu 7.1.2014.

14 Wikipedia. Network Automation. Verkkosivut. <http://en.wikipedia.org/wiki/Network_Automation>. Luettu 7.1.2014.

Ohjelmakoodi ilman tietokantarakennetta

```
Start timer session "TimerSession1".
Variables
Create a variable named "vXMLName".
Create a variable named "vXMLValue".
Create a variable named "vXMLCommand" with an initial value of
"-".
Create a variable named "vLoopNum" with an initial value of "0".
Create a variable named "vDateTime".
Create a variable named "vError".
Create a variable named "vTable" with an initial value of "0".
Create a variable named "vColumn" with an initial value of "0".
Create a variable named "vFixtureIdColumnValue" with an initial
value of "0".
Create a variable named "vLastId" with an initial value of "x".
Create a variable named "vSessionErrorMessage".
Create a variable named "vCHK_StepNum" with an initial value of
"0".
Create a variable named "vLastId_STEP_RESULT" with an initial
value of "y".
Create a variable named "vNull".
Create a variable named "vFileDateTime".
Create a variable named "vSQL_Database".
Create a variable named "vErrorLog_Folder" with an initial value
of "Error_log".
Create a variable named "vTestTable".
Create a variable named "vSeqId".
Create a variable named "vPass_FolderName" with an initial value
of "Pass_Xml_Session".
Create a variable named "vFail_FolderName" with an initial value
of "Fail_Xml_Session".
Create a variable named "vTimer" with an initial value of "0".
Create a variable named "vSQL_Status" with an initial value of
"0".
Create a variable named "vPathCase" with an initial value of
"0".
Create a variable named "vCommandDir" with an initial value of
"C:\KoneFTV2\Parsing material".
Create a variable named "vErrorSessionDir".
```

```
Execute the SQL Statement: "Select * From INI".
Setup path
Loop through the dataset "PathDataset".
Increment Variable "vPathCase" by 1
Select a following case based on the result of expression:
%vPathCase%
Case 1
Set variable vCommandDir to value "%PathDataset.Path%"
End Case
Case 2
Set variable theDirectory to value "%PathDataset.Path%"
End Case
Case 3
Set variable vErrorSessionDir to value "%PathDataset.Path%"
End Case
Case 4
Set variable vSQL_Database to value "%PathDataset.Path%"
End Case
End Select block
End Loop
Check folder
If folder "%theDirectory%\%vPass_FolderName%" does not exist
then...
Create folder "%theDirectory%\%vPass_FolderName%".
Goto label Folder exist 1.
Else
Folder exist 1:
If folder "%theDirectory%\%vFail_FolderName%" does not exist
then...
Create folder "%theDirectory%\%vFail_FolderName%".
Goto label Folder exist 2.
Else
Folder exist 2:
If folder "%theDirectory%\%vErrorLog_Folder%" does not exist
then...
Create folder "%theDirectory%\%vErrorLog_Folder%".
Goto label Folder exist 3.
Else
Folder exist 3:
error:
```

```
Loop through files in directory "%theDirectory%".  Populate variable "theFileName" with the file name.
Set variable vFileDateTime to value "%FileDateTime(theFileName)%"
If file "%theFileName%" exists (must be newer than %DateAdd( "n", -30, CStr( Now() ))%) then...
Goto label XML error.
Else
Create session:
Create session from existing XML file %theFileName%. Session "XmlSession1".  If the step causes an error, retry 2 times, pausing 500ms between attempts.  If the step still errors, goto label XML error.
Open SQL query to command variable
Execute the SQL Statement: "Select * From Commands".  Connect to datasource using the connection string
Not found loop:
If %LoopNum% >= 42 then...
Goto label END.
Else
Loop through the dataset "CommandDataset".
Set variable vXMLCommand to value "%CommandDataset.TestCommands%"
Set variable LoopNum to value "%LoopNum + 1%"
Create dataset "MyXMLNodes" of containing node values of type "XPath" at XPath location "LOG/%vXMLCommand%//*".  Session "XmlSession1".  If the step causes an error, retry 1 times, pausing 500ms between attempts.  If the step still errors, goto label command not found.
Loop all vXMLCommand elements
Read values
Loop through the dataset "MyXMLNodes".
Set variable vXMLName to value "%MyXMLNodes.Value%"
Read node value at XPath location "%MyXMLNodes.Value%" and save it into automate variable "vXMLValue". Session "XmlSession1".
".    If the step causes an error, goto label SQL Case Error.
End Case
End Select block
EndLoopCase:
End Loop
Information and go to
```

```
SQL closed:
command not found:
End Loop
Goto label Not found loop.
END:
Set variable LoopNum to value "0"
End XML session "XmlSession1".
Move file(s) from "%theFileName%" to "%theDirecto-
ry%\%vPass_FolderName%".
End Loop
Goto label StopTask.
XML error:
Xml error log
Format the current date/time as "mm/dd/yyyy h:nn:ss". Use 24
hour time format.
Set variable vError to value "%vDateTime%
%theFileName%
%vSessionErrorMessage%"
Write the data "%vError%
" to the file "%vErrorSessionDir%\Error_list.txt". File is "AN-
SI" encoded.
Move file(s) from "%theFileName%" to "%theDirecto-
ry%\%vFail_FolderName%".
Goto label error.
StopTask:
If %vSQL_Status% = '1' then...
Close SQL connection.
Goto label SQL_session checked.
Else
SQL_session checked:
Stop timer from session "TimerSession1".Store elapsed time in-
side variable "vTimer".The elapsed time is returned in number of
Minutes.
Stop this task
SQL Case Error:
SQL error log
Format the current date/time as "mm/dd/yyyy h:nn:ss". Use 24
hour time format.
Set variable vError to value "%theFileName%
%vDateTime%
Table: %vTable%
```

```
Column: %vColumn%
Value: %vXMLValue%
%vSessionErrorMessage%"
Write the data "%vError%
" to the file "%vErrorSessionDir%\SQL_error_list.txt". File is
"ANSI" encoded.
Goto label EndLoopCa
```